**SOFTWARE ENGINEERING**

**1.1 INTRODUCTION:**

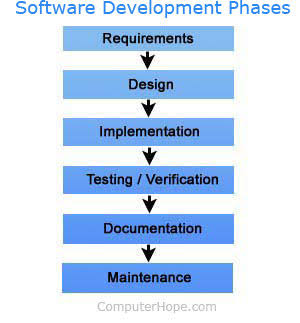
*What is software Engineering and why is software development said to be an engineering process?*

As a way of introduction, it is important that we understand what software engineering is and why software development is likened to an engineering process.

* Software engineering is the establishment and use of **sound engineering principles** in order to develop software that is **economical,** **functional, reliable and** that can work **efficiently on real machines/computer systems**.
* Software engineering can also be said to be the application of a systematic, disciplined, quantifiable approach to the development, deployment, operation and maintenance of software / an application.
* Software development is said to be an engineering process because it involves generic (progressive) developmental phases (ie the engineering principles) which start from

**Requirement specification**—> **Design—> Implementation--🡪Testing/Verification—>deployment—>Documentation -🡪 Maintenance**

just like any other engineering process. It is this generic nature of software development that makes it an engineering process is as shown in the diagram below:



**Requirement/system specification:** involves determining the exact requirement of the system. System specification is used to derive what the system should do without saying how this is to be achieved.

**Design**: this involves addressing how the system is to be implemented.

**Implementation** : this stage is the programming stage.

**System testing**: this stage is aimed at tracking bugs

**Maintenance**: this keeps the system updated for new changes that need to be implemented.

Note that the main purpose of software engineering is to **reduce risk and cost**.

**1.2 SOFTWARE DEVELOPMENT PROCESS MODELS**

Software development process is the process of dividing software development work into distinct phases mainly:

1. To improve design,
2. For project management,
3. For (product) software management.

* Software development follows a development process also known as a software development life cycle (SDLC).
* The software development cycle is therefore defined as the specific **processes or procedures or steps that are created, taken and completed by a project team in order to develop or maintain an application**.

A software “development process model" refers to a specific process chosen by a specific organization from many different approaches to modeling. Hence, a software process models is one of the software processes chosen or adjusted to give the best solution to any project.

*What is a model or what do we mean by a model in software development?*

* A software process model can be said to be an abstract representation of a software process.
* Modeling in a nutshell simply means the way a software process is represented or **presented.**
* In science, a model is a representation of an idea, an object, or a system that is used to describe or explain a phenomenon that cannot be explained directly.
* Modelling can be mathematical modelling or computer modelling.
* Computer modeling is the construction and manipulating of abstract representation of situations or natural phenomenon simulated with the help of computers.
* Computer models can be represented either mathematically or graphically (ie pictorially).
* Modelling is just the use of something or anything called models to represent a system, entity, phenomenon, process etc.
* Whereas simulation is a method/ process of implementing a model.
* The act of simulating something first requires that a model must be developed.
* A computer or software model can be an algorithm or the equations used to capture the behavior of system being modelled.
* While computer simulation is the actual running of the program that contains these equations or algorithms.
* Simulation therefore is the process of running a model. Models are used to predict what will happen in the system. It helps in making alterations and quickly seeing the outcome especially when building a prototype.

Examples software process models include:

* The waterfall model
* Prototype model
* Incremental development model
* The spiral model
* Iterative model
* The agile model
* RAD model etc

**2.1 SOFTWARE DEVELOPMENT REQUIREMENTS SPECIFICATIONS (SRS)**

* Software (system) requirements specification (SRS) is a documented concise description of a software system to be developed.
* It is a document that captures the complete description about how the system is expected to perform

* It is also known as a stakeholder requirements specification (StRS).
* These are the requirements that the ‘user/client’ is expecting from the programmer or software Engineer.
* It must be well thought out , balanced and clearly understood by all involved
* Software requirements specification lists sufficient and necessary requirements for the project development.
* To derive these requirements, the developer needs to have clear and thorough understanding of the products to be developed.
* This is achieved through detailed and continuous communications with the project team and client/user throughout the software development process.

SRS must layout the following:

* functional and non-functional requirements (*what do you understand by functional and non functional requirements?).* The functional requirements specify what the system/software should do. That is the functions that the system is expected to perform. For example: display students’ names, Depts, sex, TNU, TGP, CGPA, class of degree etc. Give examples of other functional requirements.

whereas the non functional requirements specifies how the system works or how the system should behave . For examples are the system’s attributes like the response time, throughput, scalability, utilization, reliability, maintainability, availability, usability, security, interoperability, serviceability and other system’s behavioral attributes

* may include a set of use cases that describe user interactions that the software must provide. *( what are USE CASES)*

In software engineering, a USE CASE is a **list of actions** or steps of events that shows or stipulates the various interactions that will occur or occurs between a system and user (external event). It specifies the **flow of events/interactions in a system** to achieve the desired goal. It is used to identify, clarify, and organize a set of possible sequence of interactions between a system and users in a particular environment to achieve a particular goal.

* Software requirements specification establishes the basis for an agreement between users and software developer on how the software product should function.
* It is a rigorous assessment of requirements before the more specific system design stages,
* Its goal is to reduce later redesign.
* It should also provide a realistic basis for estimating software costs, risks, and schedules.
* When software requirements are specified appropriately, it can help prevent software project failure.

**2.1b Qualities of SRS**

* Mustbe correct
* Complete
* Unambiguous
* Consistent
* Verifiable
* Modifiable
* Traceable
* Stable

**Software Development Methodology**

What is software development methodology?

Definition: In [software engineering](http://en.wikipedia.org/wiki/Software_engineering) , Software design methodology is the **sum total of the methods (tactics, strategies, policies, devices, outline, structure, procedures, style etc ) needed or used by software engineer** in the design of software.

That is to say:

In [software engineering](http://en.wikipedia.org/wiki/Software_engineering), software design methodology can be

* a framework
* or an **approach**

used for the development of software application

### Software Development Methodology as a Framework

What is a *framework*?

Framework is a kind of laid out structure for software DevOps (Development and Operations).

Now what is DevOps?....

DevOps as used in software Engineering is a kind of combining software development with Information technology operations so as to shorten the system development lifecycle.

Hence we can say that software development methodology is a framework that is used;

* to structure, plan, and control the [process of developing](http://en.wikipedia.org/wiki/Software_development_process)  a software or an information system.
* Framework is a foundation or an incomplete structure which can be extended to build an information system.
* It’s a universal reusable software environment that provides some functionalities to facilitate the dev of a software.
* It may include code libraries, compilers, other tools needed in software development process.
* It is a kind of scaffold / platform for developing software products.
* It streamlines the software development such that programmers don’t have to write all codes each time.
* is used to express, show, or describe the structure, plan, and control the [process of developing](http://en.wikipedia.org/wiki/Software_development_process) an [information system](http://en.wikipedia.org/wiki/Information_system) .
* It is a kind of loose but incomplete structure which still gives room for other practices and tools needed to develop a software. That is it can be extended to develop a software product.
* Examples of software frameworks include: waterfall, spiral , prototype methodologies.

### Software Development Methodology as an Approach

Software development methodology may also refer to the approach used or applied in order to actualize an information system. This approach applied here is also called software Development Life Cycle (SDLC). In general software methodology approaches include:

* SDM (System Development Methodology)
* [Structured Systems Analysis and Design Method](http://en.wikipedia.org/wiki/Structured_systems_analysis_and_design_method)ology (SSADM)
* [Information Requirement Analysis/Soft systems methodology](http://en.wikipedia.org/wiki/Soft_systems_methodology)
* [Object Oriented and Design Methodology (OODM)](http://en.wikipedia.org/wiki/Object-oriented_programming)
* [Rapid Application Development](http://en.wikipedia.org/wiki/Rapid_application_development) (RAD)
* [Dynamic Systems Development Method](http://en.wikipedia.org/wiki/Dynamic_systems_development_method)ology (DSDM)
* [Scrum](http://en.wikipedia.org/wiki/Scrum_(development))
* Component-based methodology
* knowledge-based methodology
* Experimental methodology
* Agent-based methodology
* Formal methods
* Prototyping
* Extreme programming

**Hence,** software development approaches that have been used since the origin of information technology are **broadly classified into two** as given below:

* Traditional Software development methodology approach
* Agile methodology approach ( popularly called Agile manifesto)

***Traditional Software Development Methodology***:

The traditional software development methodology or the oldest formalized software methodology approach for building [information systems](http://en.wikipedia.org/wiki/Information_system). The main idea of the SDLC has been "to pursue the development of information systems in a very deliberate, structured and methodical way (cycle) requiring each stage of the [life cycle](http://en.wikipedia.org/wiki/Software_development_process) from inception of the idea to delivery of the final system to be carried out rigidly and sequentially.

Examples are:

* + Waterfall: a linear framework
  + Spiral: a combined linear-iterative framework
  + Incremental: a combined linear-iterative framework or V Model
  + Prototyping: an iterative framework
  + Rapid application development (RAD): an iterative framework

***The*** traditional methodologies use linear approach where the stages of the software development process must be completed in a sequential order. This means one stage must be completed before the next one begins.

***The Agile methodology Approach***

Agile is an umbrella term for several **iterative and incremental** software development approaches, where each of the variations can be on its own. This is called the [Agile Manifesto](http://en.wikipedia.org/wiki/Agile_Manifesto), and was formulated in 2001. **It uses iterative and team based approach**. The main objective of Agile methodology is to

* **quickly deliver the application that has complete and functional components**.
* That instead of completing software development in a sequential order, they are developed in bits called sprints.
* It takes an average of four weeks to complete a sprint. Different stages of software development life cycle can be revisited when need arises.

Examples of Agile approach to software development involve the following stages: **project initiation, sprint planning and demos**.

Examples of methodologies under Agile manifesto are but not limited to:

* + Scrum
  + Extreme programming
  + Adaptive Software Development (ASD)
  + Dynamic System Development Method (DSDM)

**The Traditional Software Methodology vs the Agile Methodology.**

The major difference between the two approaches is in the sequence of phases taken to complete software project.

* The traditional uses a linear approach while Agile uses an iterative and team based approach.
* Instead of developing task in sequence the Agile complete in sprints usually within one to four weeks.
* The traditional is rigid while agile is flexible
* This means making unplanned changes in the software is costlier while in agile changes can easily be made even after the project has be completed.
* The traditional method show details of entire project before the project starts.
* The traditional only involve customers at the early stage of the software dev, may be during requirement specification and will only see the software again at completion.
* But in agile methodology customers are highly involved in every stage and can make suggestions for improvement.
* The traditional is highly documented but not so with agile due quick time delivery.

**Factors to Consider in Choice of Software Development Methodology**

1. Schedule Time/ Speed needed to complete the project
2. Size of the project
3. Cost/ Fund /resources available.
4. Level of collaboration and interaction that exist among the

software project team

1. Quality of product needed.

Hence we can say that whenever time is a constraint….Agile should be an option. This is because Agile allows you to quickly release a product which you can later modify to add more features.

With limited resources too…Agile comes to mind. Agile works best if we need a basic software to keep your business running and going.

Agile provides more benefits for start ups. Agile also fits in for small to medium size applications. More customer involvement changes are constantly made to the application and as such gives a better quality.

On the other hand:

For large enterprise projects… Consider traditional methodology. Here the specification and requirements must be clearly specified or defined before the project begins. Also…. when the project team (software developers, tester, vendors, designers, customers) are not in staying together in a place Agile become difficult… hence traditional is an option.

**Note:**

A good software project should be developed on time , within budget and to the satisfaction of the customers irrespective of the approach.

In the course of this lecture, we aregoing to consider the following methodologies:

1. Object oriented methodology
2. Component-based methodology
3. knowledge-based methodology
4. Experimental methodology
5. Agent-based methodology
6. Formal methods,
7. prototyping

**4.1 Object Oriented and Design Methodology**

Object oriented methodology is an approach to building applications/systems by using the **object oriented paradigm** throughout the entire software development life cycle.

**Objects** in this context are real life things like cars, humans, trees houses etc.

**Paradigm** means a model, example, architype, prototype of something

Object oriented paradigm therefore means modelling based on objects only. In OOP, the following terms are frequently used: **properties, methods, events, classes, instances**.

These Objects are known to possess some attributes/characteristics size, shape, color, height, and these are called **properties”**

“**Methods**” are procedures used to modify the objects.**(**e.g., drive, stop, start, an object such as car)

**Events** are the actions taken on the system to carry out the methods. **e.g., Clicking , double Clicking the Mouse**

**Classes** are like a group name for group of objects, a particular structure for representing a particular group of objects. E.g., Car is a class; Polo, Mini, Beetle, are objects.

**“ A class thus is a blueprint or prototype , that defines the variables and the methods common to all objects of that class.** A class can define types of operations, or methods, that can be performed on an object. For example, the car class might specify an accelerate (method) which would update the speed (attribute) of the car object. A class is therefore used in organizing data into different data types so a programmer can reuse elements when making multiple instances of that data type

**Instances** depict the different types of classes.  In the above example, there are three instances:**polo, mini , and beetle** .

In object Oriented Methodology (OOM), we have

Object Oriented Design (OOD) --- used for design

Object Oriented Analysis (OOA) ----- used for analysis

Hence combining both the design and analysis phases we have what we call Object Oriented Analysis and Design Methodology (OOADM).

**Characteristics of OODM:**

1. It is an iterative and incremental software development process
2. It encourages and facilitates the re-use of software components.
3. Ensuring higher productivity,
4. lower maintenance cost
5. It produces better quality software product.
6. It employs international unified modeling language (UML) from the object management group. UML is the widely accepted language for OOA and OOD.

Software development using Object oriented methodology undergoes six phases namely:

* 1. planning stage
  2. architecture definition stage
  3. technical architecture definition stage
  4. the incremental delivery planning stage
  5. the incremental design and build stage
  6. the deployment stage.

\*Object oriented methodology, is developed on a component basis by sharing of its other system components.

The OOM gives the software engineer the opportunity to determine the followings:

* What the objects of the system are
* What responsibilities and relationships an object has with the other objects
* How these objects behave over time

**Objectives of Object Oriented Methodologies**

* To encourage greater re-use of software components
* To produce a more detailed specification of system constraints.
* To have fewer problems with validation (Are we building the right product?).

**Benefits of Object Oriented Methodologies**

1. It represents the problem domain, because it is easier to produce and understand designs.
2. It allows changes more easily.
3. It provides nice structures for thinking, abstracting and leads to modular design.
4. *Simplicity*: The software object's model complexity is reduced and the program structure is very clear.
5. *Reusability*: It is a desired goal of all development process. It contains both data and functions which act on data. It makes easy to reuse the code in a new system. Messages provide a predefined interface to an object's data and functionality.
6. *Increased Quality*: This feature of increase in quality is largely a by-product of this program reuse.
7. *Maintainable*: The OOP method makes code more maintainable. The objects can be maintained separately, making locating and fixing problems easier.
8. *Scalable*: The object oriented applications are more scalable than structured approach. It makes easy to replace the old and aging code with faster algorithms and newer technology.
9. *Modularity*: The OOD systems are easier to modify. It can be altered in fundamental ways without ever breaking up since changes are neatly encapsulated.
10. *Modifiability*: It is easy to make minor changes in the data representation or the procedures in an object oriented program.
11. *Client/Server Architecture*: It involves the transmission of messages back and forth over a network.

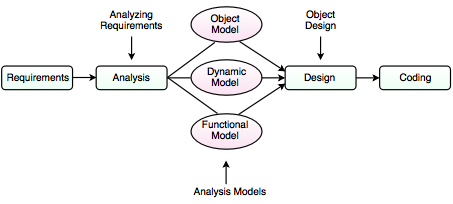
**Types / Generations of Object Oriented Methodologies**

We have three types/ generations of Object Oriented Methodologies

* Object Modeling Techniques (OMT)
* Object Process Methodology (OPM)
* Rational Unified Process (RUP)

**Object Modeling Techniques (OMT)**

OMT is among the first generation object oriented methodologies introduced as early as 1991. It uses three different models (the object model, Dynamic model, and functional model) as shown in the diagram below.



The main goal of the analysis is to build real world models using the available user requirements and developers /managers information.

The **Object Model** depicts the object class and their relationships as a class diagram, which represents the static structure of the system. It observes all the objects as static and does not pay any attention to their dynamic nature.

While the **Dynamic Model** captures the behavior of the system over time and the flow control and events in the Event-Trace Diagrams and State Transition Diagrams. It portrays the changes occurring in the states of various objects with the events that might occur in the system.

The **Functional Model** describes the flow of data and the changes that occur to the data throughout the system.

The **Design Stage** specifies all of the details needed to describe how the system will be implemented. In this phase, the details of the system analysis and system design are implemented. And the objects identified in the system design phase are designed.

**2. Object Process Methodology (OPM)**

This is also called as second generation object oriented methodology and was first introduced in 1995. It has only one diagram that is the Object Process Diagram (OPD) which is *used for modeling the structure, function and behavior of the system.* It has a strong emphasis on modeling but has a weaker emphasis on process. It consists of three main processes:

* Initiating: It determines high level requirements, the scope of the system and the resources that will be required.
* Developing: It involves the detailed analysis, design and implementation of the system.
* Deploying: It introduces the system to the user and subsequent maintenance of the system.

**3. Rational Unified Process (RUP)**

This third generation was developed in Rational Corporation in 1998. It consists of four phases which can be broken down into iterations.

* Inception
* Elaboration
* Construction
* Transition

Each iteration consists of nine work areas called disciplines. A discipline depends on the phase in which the iteration is taking place. For each discipline, RUP defines a set of responsibilities of the members of the development team).

**In OOM we use** Object Oriented Programming

**OBJECT ORIENTED PROGRAMMING** is a programming language in which pgms are organized /arranged around data (called objects) rather than functions and logic.Before now programing was seen as how to write logic not how to define data (objects).But OOP takes a different view. OOP only cares about the data (objects) it wants to manipulate rather than the logic /computations required to manipulations**.** Note in OOP, that objects are just data fields that have unique attributes and behavior.

And we said that examples of objects are physical entities such as human beings ( attributes such as name, address, size etc),cats, dogs or even houses, buildings, ( with attributes like height, colour, shape ) or computer Widgets on computer desk tops like mouse buttons, scroll bars. OOp is opposed to the traditional approach to programming where emphasis is on how the logic is written rather than how to define the data within the logic. In oop the first task of a developer is to identify the objects and their relationship with one another. Simply put , OOP is used to manipulate data fields (objects).

Note a group of objects is called a class. A class of objects defines the type of data it contains and the sequence of logic ( called methods) to manipulate them. That is each logic sequence /procedure/ operation on any object or class of objects is called a **method.** **Examples** OOP pgming languages are: Java, java script, python, VB.Net, PHP scala, ruby. **OOP** is well suited for projects that are large, complex and are frequently maintained or updated. It is applicable in collaborative software development. **Major benefits of OOP** are scalability, reusability and efficiency.

**Principles of OOP**

1. **Encapsulation : this** principle is a kind of class self-privacy. That is each class maintains its own privacy from classes, other objects outside this class have no access to interfere with another class. But they can use make use of call list declared as publics functions/ methods/variable. This is equally called data hiding and provides security to the software.

**As an eg in java we can hv:**

**Class Account{**

**Private** int account-number;

**Private** int account-balance;

**Public** void show Data(){

//code to show data

}

Public void deposit(int a){

1. **Abstraction: just like the** above principle, only internal mechanisms relevant to outsiders are revealed while hiding other private/unnecessary implementations. That is hiding implementation details from the user.
2. **Inheritance:** the concept **class or data class in OOP** makes it possible to have or define subclasses of data objects that share ( ie inherit) all or some of the characteristics of the main class. Hence the ability of the subclasses to share some of the characteristics of the main class is called inheritance. This principle allows the software developer to re-use a common logic while still maintaining a unique hierarchy. That is a relationship between two or more classes where derived or sub classes inherits the behavior and attributes of the existing classes. This reduces time and ensures high level of accuracy. It is intended to help re use of existing codes with little or no modification.
3. **Polymorphism:** this principle allows the software developer to create any new data that is not already defined in the programming language. That is to say objects can take more than one form depending on the context. This also implies that the programmer determines the meaning and usage of objects depending on the context.

**PROBLEMS OF OOP**

* OOP is not easily understood by beginners
* OOP over emphasizes on data component of software development and does not give enough attention to the computation or algorithm
* OOP codes are more complicated and take longer time to compile.

**4.2 KNOWLEDGE-BASED METHODOLOGY**

**A knowledge base** is just a repository or library of information about something like product, a topic, department, organization…anything at all that is designed to provide self served help… this is to free up users time which often is better used in challenging and complex scenarios.

**Knowledge based methodology** or knowledge based engineering is simply the application of knowledge based systems technology in the design of software products.

**What are knowledge based systems?**

Knowledge based systems are just **computer programs** that reason and use a knowledge base to solve complex problems. A knowledge base is a repository (warehouse/storehouse) of knowledge just like we have database. A knowledge base system is a technology used to store complex structured and unstructured information used by computer systems to solve problems. Knowledge based systems unlike other systems represent the **stored knowledge** vividly using rules and tools like **ontologies** (means a representation or formal naming/definitions used in defining the categories, properties and relations between the given concepts data or entities) rather using codes as in a normal computer program.

**Components of knowledge based system**

* The knowledge
* The user interface
* The Inference engine

***Knowledge*** here represents real world facts / information gotten or gathered from real world experts from the particular field of interest. Example is collection of information in a medical field—for collection of information in a given field -- medical diagnosis, for example medical diagnosis.

***User interface*** represents where the interactions between humans and computers occur. Users query and interact with system through this interface.

***Inference engine*** represents logical assertions / rules and conditions that are used for decision making. The inference engine deduces insights from the information housed in the knowledge base.

In Summary, a knowledge-based system (KBS) captures the knowledge of human experts to support decision-making. They exhibit a kind of intelligence called artificial intelligence. Examples of knowledge-based systems include expert systems, which are so called because of their reliance on human expertise.

**Knowledge based methodology**

In a knowledge-based system, the methodology may vary with respect to its problem-solving method or approach. For example, some systems encode expert knowledge as rules and are therefore referred to as rule based methodology or **rule-based systems**. Another approach, case-based reasoning, substitutes cases for rules, hence called **case based methodology or cased based systems** .Cases are essentially solutions to existing problems that a case-based system will attempt to apply to a new problem. Hence, Knowledge-based systems use **a rules-based or case-based approach**. Examples

* MYCIN, was an early knowledge-based system created to help doctors diagnose diseases. Healthcare has remained an important market for knowledge-based systems, which are now referred to as clinical decision-support systems in the health sciences context.
* Early AI used "**top-down approach.** Examples of AI following the bottom up approach include **neural network systems**, a type of deep-learning technology that concentrates on signal processing and pattern recognition problems such as facial recognition.
  1. **Agent-Based Methodology**

Agent-based Software engineering also known as Agent Oriented Software Engineering (AOSE) is a software development methodology that is employed in development of **software agents** . A software agent is a **computer program** that acts for/ on behalf of the user. Software agents are computer applications that are designed to autonomously sense and respond to environment in pursuit of certain designed objectives.These software agents exhibit some kind of intelligence, hence are also known as **intelligent agents.** An intelligent system is a system that can sense and respond to its environment in pursuing its goals and objectives. It learns and adapts based on past experiences. They can be used to assist the user in performing repetitive tasks such as seeking information, shopping , scheduling, monitoring, control, negotiation, and bargaining.That is there is an agreement for the program to act or represent the user in an established relationship/interaction. It may include the ability to take decisions on behalf of the user. Agents are **autonomous ( meaning : independent, self-sufficient) software units** used in decision making and exchange of information.

Note: a system is autonomous:

* If it requires little help from the human user
* If we don’t have to tell it what to do step by step
* If it can set its own goal and the way to actualize it
* If its behavior is determined by its own experience
* If we don’t understand its internal workings.

These software agents are known as **bots and they come in various shapes and forms** examples may include **knowbots , taskbots**, **Robots, Chatbots, shopbots, softbots, personal agents, information agents,** etc. These agents are abstractions of living /autonomous entities of real world. It is this concept that makes modeling and implementing software systems with autonomous behavior, decision making, easy and realistic. An agent can be single (individual) agent or multiple agent. But no matter the form, intelligent agents exhibit one or more of the following characteristics.

**Characteristics of software agents**:

**Autonomy**: They can stand on their own.

**Intelligence**: being able to exhibit intelligent behavior such as reasoning, generalizing, learning, dealing with uncertainty, using heuristic and natural language processing.

**Adaptive learning**: being able to learn and adapt to their external environment.

**Social ability**: able to communicate , collaborate, bargain, compete, and interact with other agents and humans on behalf of their owners. Also able to communicate with using agent communication language ACL. Example is an autonomous pilot controlling an aircraft.( auto pilot).

**Make decisions** about what to do without direct intervention of humans or other agents.

**Be reactive**: they are situated in an environment (which may be physical world, user interface, internet) where they will be able to perceive their environment through sensors and respond timely to changes that may occur.

**Pro-active**: agents do not only act in response to their environment, they have ability to take initiative.

**Have human like qualities** like natural language

**Human like understanding**- take initiative.

**Mobile** – being able migrate to themselves from one system to another in a network such as web.

**Goal oriented**- able to act in accordance with the built in goals and objectives.

* **Agent based system are built around three important methodologies**:

1. Logic ---- for reasoning
2. Game theory----for strategies
3. Algorithms----- for artificial agents.

**Agent based systems in mainly used in**

* artificial intelligence,
* neural networks, (a computer system modelled on human brain and nervous system)
* Robotics etc.
  1. **Experimental-based methodology**

Experimental software engineering all about gathering evidence (data) through measurements and experiments needed when developing software systems. This evidence / data is used as basis of theories about the processes involved in software engineering.

**Empirical software engineering** is a related concept, sometimes used synonymously with experimental software engineering. Empirical software engineering emphasizes the use of empirical studies of all kinds to accumulate knowledge. **Empirical research** is based on firm **verifiable data** collected by observation of facts under natural condition or obtained through experimentation.

**Advantages of experimental methodology**

1. Experimental methodology are valuable tools for software engineers who are involved in evaluating and choosing between different methods, techniques, languages, and tools.
2. The purpose of experimentation in software engineering is to introduce students, researchers to empirical studies in software engineering using controlled experiments.
3. The introduction to experimentation is provided through a process perspective and the focus is on the steps that we have to go through to perform an experiment.
   1. **COMPONENT-BASED METHODOLOGY**

This is a software development methodology that emphasizes on the development of software systems with the help of **reusable** software components. Reusability is an important characteristic of high quality software components. By this it mean that programmer should design and implement software components in such a way that many different programs can reuse them. Reusability means:

* Fully documented
* Thoroughly tested
* Robust—has comprehensive input validity checking technique
* Able to pass back appropriate error messages or return codes
* Designed with the awareness that it put to unforeseen uses.

A **component** is a software unit whose functionality and dependencies are completely defined by its interfaces. A software component is a software package , or a web service, web resource, a module that combines or compresses a set of related functions. That is to say that all system processes separated into different components contains data and functions that are semantically related. Components communicate with each other via interfaces. A component specifies the service it can offer to other components and vice versa. During the CBSE process, the processes of requirements engineering and system design are interleaved.

Component composition is the process of ‘wiring’ components together to create a system.

When composing reusable components, **adaptors** are written to reconcile different component interfaces. In choosing compositions, you have to consider required functionality, nonfunctional requirements and system evolution.

Component-based methodology is applicable to Component Based Software Engineering (CBSE). It is also known as component-based Development (CBD).

***What do we mean by component here ?***

* Component here means software component.
* A software component is an independent executable entity that can be made up of one or more executable objects.
* It is a software element that conforms to a component model and can be independently deployed and composed without modification according to a composition standard.
* That is to say, a software component also has specified interfaces that can be deployed independently.
* The component is an independent, executable entity. It does not have to be compiled before it is used with other components.

A **component model** is a definition of standards for component implementation, documentation and deployment. The component model specifies how interfaces should be defined and the elements that should be included in an interface definition. It is a definition of the properties that components must satisfy, the methods and mechanisms for component composition. Examples of component models

* EJB model --- (Enterprise Java Beans)
* COM --- component object model
* .NET model
* CORBA ---- Component object request broker architecture

**Component composition**: The process of assembling components to create a system is called component composition. Composition involves integrating components with each other and with the component infrastructure. Normally‘glue codes’ are written to integrate components.

**Types of component composition**

* **Sequential composition:** where the composed components are executed in sequence.
* **Hierarchical composition**: where one component calls on the services of another.
* **Additive composition:** where the interfaces of two components are put together to create a new component.

In summary we can say that **Components are:**

**Standardized:** Component standardization means that a component that is used in a CBSE process has to conform to a standardized component model.

**Independent:** A component should be independent – it should be possible to compose and deploy it without having to use other specific components.

**Composable:** For a component to be composable, all external interactions must take place through publicly defined interfaces. In addition, it must provide external access to information about itself such as its methods and attributes.

**Deployable** : For component to be deployable, a component has to be self-contained and must be able to operate as a stand-alone entity on some component platform that implements the component model. This usually means that the component is a binary component that does not have to be compiled before it is deployed.

**Documented**: Components have to be fully documented so that potential users of the component can decide whether or not they meet their needs. All the syntaxes and semantics of all component interfaces have to be specified.

**Advantages of CBSE**

1. Component-based software engineering (CBSE) is an approach to software development that relies on **software reuse**. It emerged from the failure of object-oriented development to support effective reuse. In object oriented development single object classes are too detailed and specific. But Components are more abstract than object classes and can be considered to be standalone service providers.
2. Apart from the benefits of reuse, CBSE is based on sound software engineering design principles: Components are independent so do not interfere with each other; Component implementations are hidden. Communication is through well-defined interfaces.
3. Component platforms are shared and reduce development costs.
4. Components are language independent.
   1. **Software Engineering Formal methods**

In software engineering, formal methods are specific **mathematical approaches** to solving both software and hardware problems at the requirement, and specification and design levels of development process. These formal methods are mostly employed for safety or security purposes. Formal methods models are concerned with the application of **mathematical techniques** in designing and implementing the software. It lays the foundation on which the system is based. This eliminates errors or mistakes which are difficult to overcome using other models. The use of formal methods for software and hardware design is motivated by the expectation that, as in other engineering disciplines, performing appropriate mathematical analysis can contribute to **the reliability and robustness of a design**. Formal methods comprise formal specification using mathematics to specify the desired properties of the system.

Advantages of formal methods include:

* It helps to discover ambiguity, incompleteness and inconsistencies in the software.
* Gives error free software
* It grows an effective solution incrementally at every iteration
* It does not involve high complexity rate.
* Formal language can verify any \*\*self consistency.

Disadvantages:

* Its time consuming
* It is difficult to use this as a communication mechanism for those who are not technical
* Requires extensive training for developers to understand the model
  1. **Prototyping Methodology**

The Prototyping Model is a systems development method in which a prototype (an approximation of a final system or product) is built, tested, and then reworked as necessary until an acceptable prototype is finally achieved from which the complete system or product can now be developed. Software prototype is simulated version of the intended software product. Prototype provides **initial look and feel** of the software product and simulates few aspect of actual product. This model works best in scenarios where not all of the project requirements are known in detail ahead of time. It is **an iterative, trial-and-error process** that involves both the developers and the users.

There are several steps in the Prototyping Model:

* The new system requirements are defined in as much detail as possible. This usually involves interviewing a number of users representing all the departments or aspects of the existing system.
* A preliminary design is created for the new system.
* A first prototype of the new system is constructed from the preliminary design. This is usually a scaled-down system, and represents an approximation of the characteristics of the final product.
* The users thoroughly **evaluate the first prototype, noting its strengths and weaknesses**, what needs to be added, and what should to be removed. The developer collects and analyzes the remarks from the users.
* The **first prototype is modified, based on the comments supplied by the users**, and a second prototype of the new system is constructed.
* The second prototype is evaluated in the same manner as was the first prototype.
* The preceding steps are iterated/ repeated as many times as necessary, until the users are satisfied that the prototype represents the final product desired.
* The final system is constructed, based on the final prototype.
* The final system is thoroughly evaluated and tested.
* Routine maintenance is carried out on a continuous basis to prevent large-scale failures and to minimize downtime.